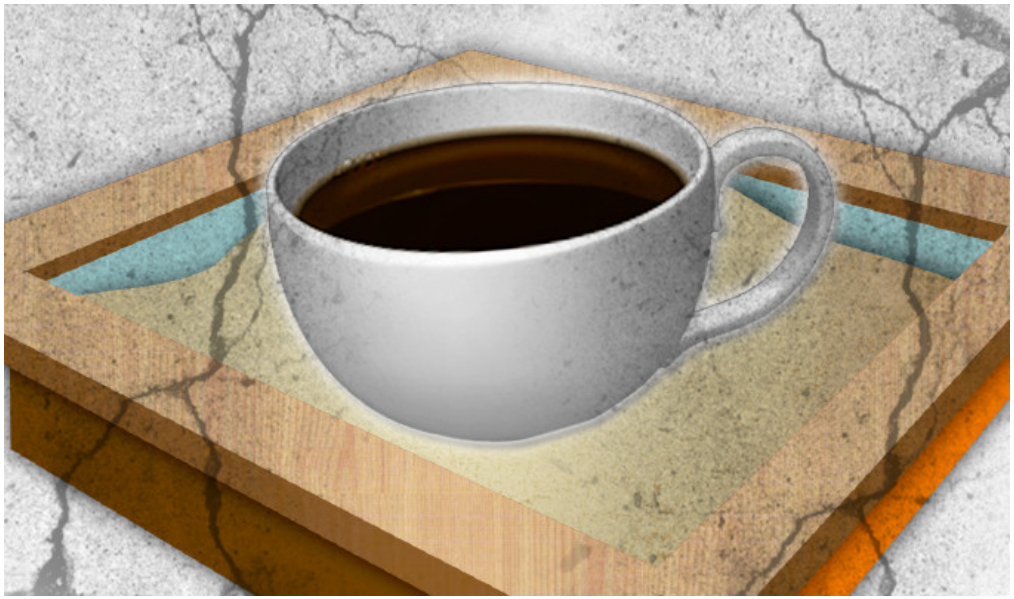


Побег из песочницы. Уязвимости нулевого дня в Java

Михаил Дударев

Licel, 2014

Jpoint

О нас

- Михаил Дударев, работает с Java Security более 15 лет, основатель проекта jCardSim, Java Card симулятора, завоевавшего награду Duke's Choice Award 2013, со-основатель компании Licel.

- Licel занимается созданием решений для защиты программного от нелегального использования и модификации третьими лицами для Java и Android платформ.

Краткое содержание

- Архитектура безопасности Java
- Побег из песочницы
  - CVE-2012-0507 (Carberp)
  - CVE-2012-1723
- Разбор malware на базе CVE-2012-1723
- Развитие механизмов безопасности Java

Самая первая уязвимость Java

Февраль 1996

“The Java DNS Security Bug (CERT CA-1996-05)”

Платформы: Java 1.0/Netscape Navigator 2.0

Описание: *При проверке возможности открытия апплетом сетевого соединения, считалось что все IP-адреса, указанные в ответе DNS-сервера являются доверенными.*

Архитектура безопасности Java

- Система привилегий
- Область применения
- Основные элементы

Система привилегий

- Регламентирует доступ участка кода к различным критическим ресурсам
  - Ввод/вывод, запуск внешний приложений и загрузка нативных библиотек, управление средой выполнениния
- Разные части кода одного приложения могут выполняться с различными правами доступа

Система привилегий

- Разграничение привилегий
  - Каждый компонент обладает минимальными необходимыми привилегиями для его работы
  - Компонент не может выполнять другие привилегированные операции, кроме предусмотренных

Система привилегий

- Разделение ответственности
  - Компоненты, обладающие привилегиями должны ограничивать доверие к другим компонентам
  - Необходимо контролировать взаимодействие между компонентами с разными уровнями привилегий

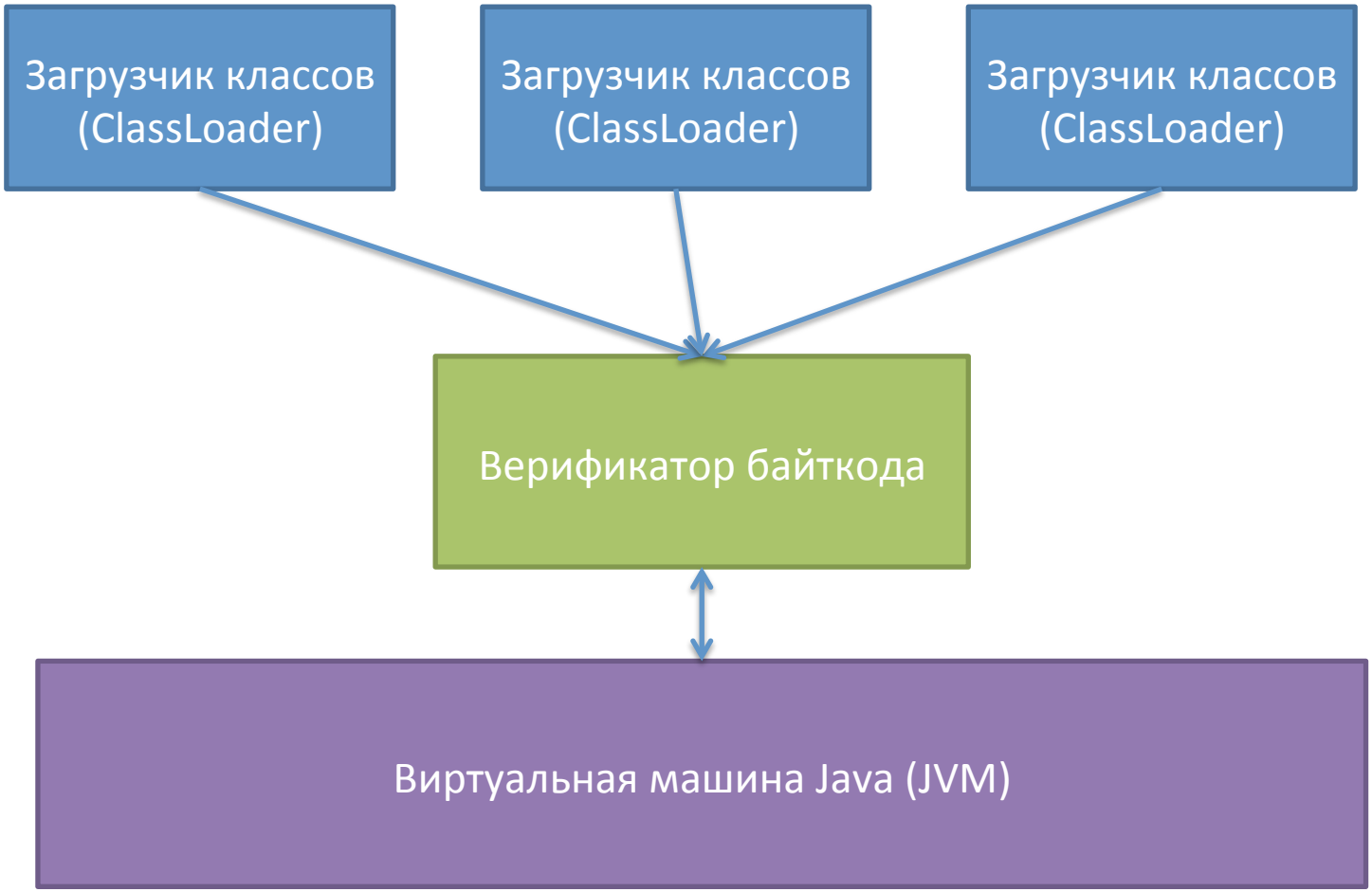
Область применения

- Апплеты (Applets)
- Java Web Start
- JavaFX
- Сервлеты (Servlets)
- Сервера приложений (Application Servers)
- OSGI-контейнеры

Основные элементы

- JVM security
  - Загрузчики классов (ClassLoader)
  - Верификатор байткода
  - Внутренние механизмы безопасности
- Java security (API)
  - Контроль доступа (SecurityManager)
  - Происхождение (CodeBase + Signature)

JVM Security



Загрузчик классов (ClassLoader)

- Первая линия обороны
- Отвечают за загрузку классов, необходимых для работы
- Каждый экземпляр обладает уникальным пространством имен
- Имеют иерархическую структуру наследования

Верификатор байткода

- Проверяет
  - Корректность формата байткода
  - Корректность определения класса
  - Код не нарушает прав доступа (private/ protected..)
  - Совместимость объектов по типу

Верификатор байткода

- Гарантирует
  - Отсутствие stack overflows/underflows
  - Корректность использования и установки локальных переменных
  - Корректность параметров инструкций байткода
  - Доступ на основании модификаторов (public/ private/protected)

Верификатор байткода

- Как работает (Очень упрощенная модель)
  - Проверяем операнды инструкции
  - Моделируем исполнение инструкции
  - Вычисляем новое состояние (количество регистров, глубина стека, типы значений в регистрах и стеке)
  - Применяем вычисленное состояния к предыдущим верифицированным инструкциям
  - Объединяем новое состояние с предыдущим
  - Ищем возникшие ошибки

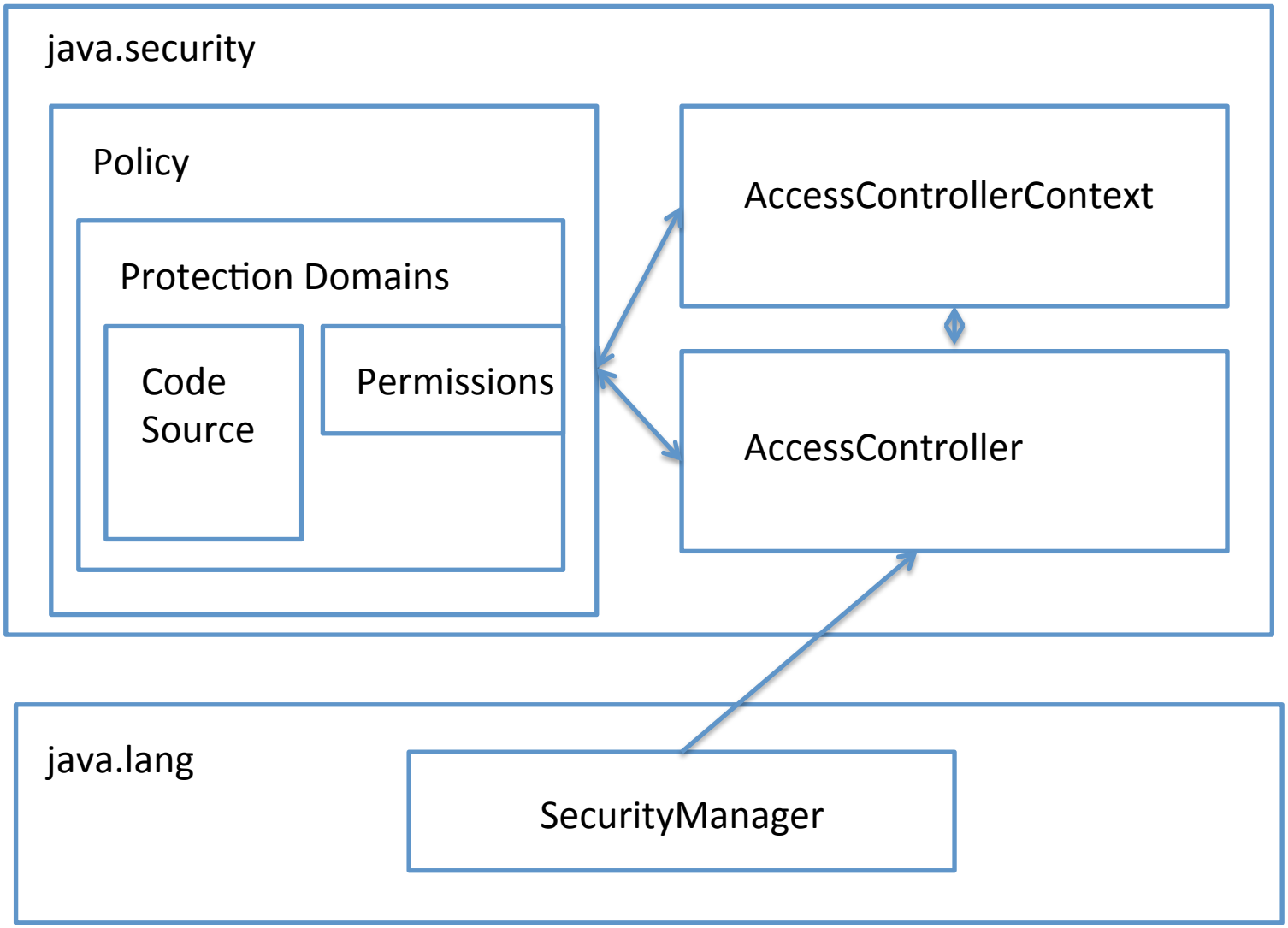
Где сработает верификатор ?

```
.method public static main([Ljava/lang/String;)V
    bipush 100
    invokevirtual java/lang/Object/clone()Ljava/lang/
Object;
    return
.end method
```

А здесь ?

```
.method public example()V
    .limit locals 2
    .limit stack 10
    iconst_4
    istore_1
  Loop:
    aconst_null
    iinc 1 -1
    iload_1
    ifne Loop
    return
.end method
```

Java security (API)



Менеджер безопасности (SecurityManager)

- Реализует политики безопасности
- Всегда вызывается перед выполнением потенциально опасной операции
- Содержит необходимые check* методы для проверки доступности операции
- Отвечает за реализацию песочницы для апплетов

Менеджер безопасности
Что под капотом ?

- AccessControllerContext
  - Содержит текущую политику безопасности
  - Поле Thread private inheritedAccessControlContext
- AccessController
  - Реализация методов check* из SecurityManager
  - doPrivileged() – выполнения кода, требующего привилегий

Пример политики

Все разрешено

```
grant codeBase "file:${{java.ext.dirs}}/*" {
        permission java.security.AllPermission;
};
// MacOS X extensions START
grant codeBase "file:${user.home}/Library/Java/Extensions/*" {
        permission java.security.AllPermission;
};
grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See the API specification of java.lang.Thread.stop() for more
        // information.
        permission java.lang.RuntimePermission "stopThread";
        // allows anyone to listen on un-privileged ports
        permission java.net.SocketPermission "localhost:1024-", "listen";
        // "standard" properies that can be read by anyone
        permission java.util.PropertyPermission "java.version", "read";
           …
}
```

Пример использования

```
public void connect(SocketAddress endpoint, int timeout) throws IOException {

    InetSocketAddress epoint = (InetSocketAddress) endpoint;
    SecurityManager security = System.getSecurityManager();

    if (security != null) {
        if (epoint.isUnresolved()) {
                security.checkConnect(epoint.getHostName(),epoint.getPort());
         } else {
                security.checkConnect(epoint.getAddress().getHostAddress(),
                                      epoint.getPort());
        }
     }
}
```

Установлен ли SecurityManager

Проверка прав

Пример использования

```
public InputStream getInputStream() throws IOException {
        if (isClosed())
            throw new SocketException("Socket is closed");
        if (!isConnected())
            throw new SocketException("Socket is not connected");
        if (isInputShutdown())
            throw new SocketException("Socket input is shutdown");
        final Socket s = this;
        InputStream is = null;
        try {
            is = (InputStream)
                AccessController.doPrivileged(new PrivilegedExceptionAction() {
                    public Object run() throws IOException {
                        return impl.getInputStream();
                    }
                });
        } catch (java.security.PrivilegedActionException e) {
            throw (IOException) e.getException();
        }
        return is;
}
```

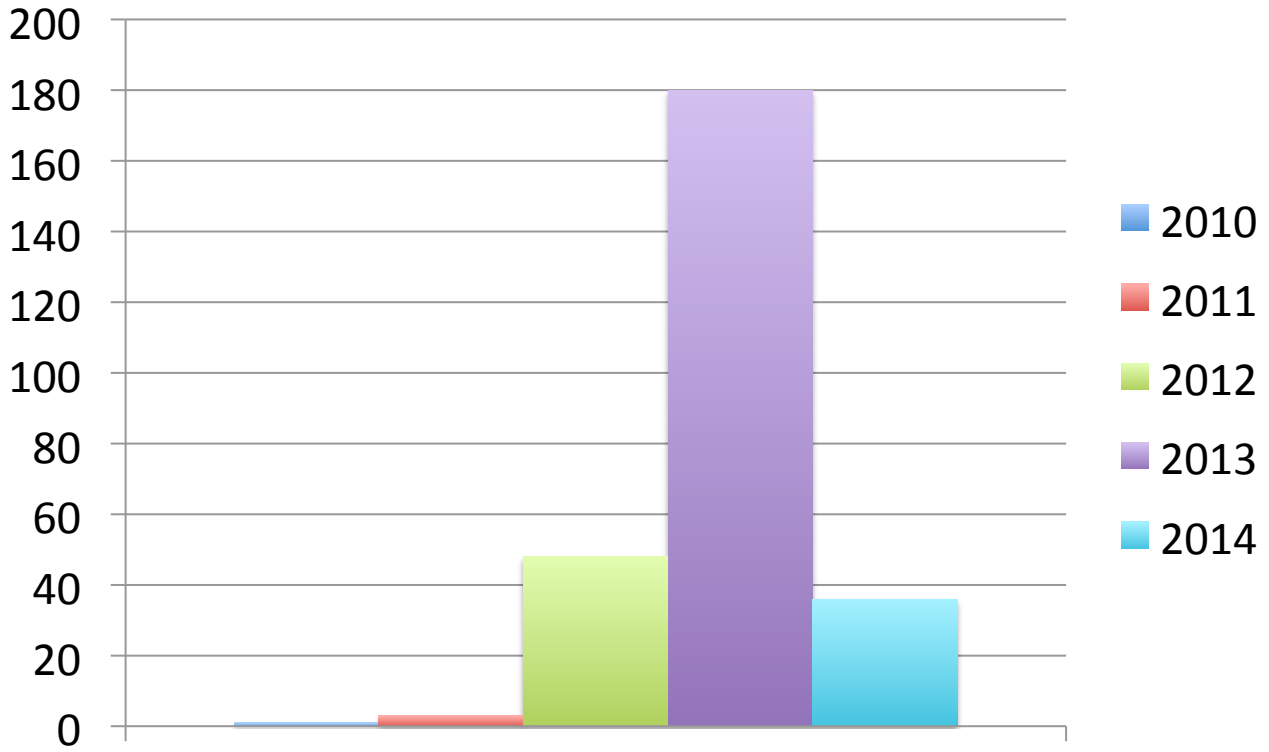
Привилегированный код

Побег из песочницы

Побег из песочницы

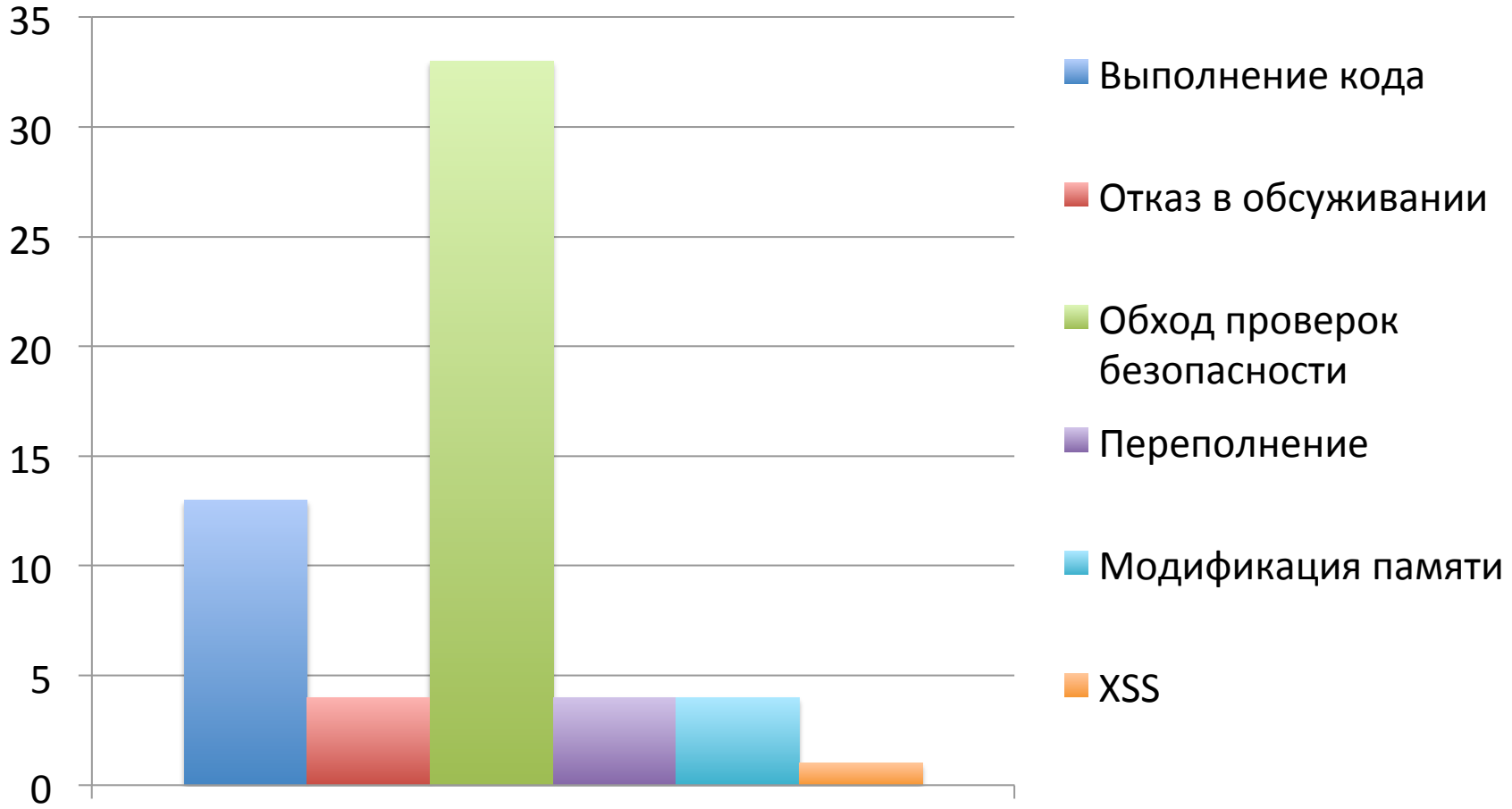
- Основные векторы атаки
  - Повышение привилегий недоверенного кода
  - Отключение SecurityManager
- Уязвимые компоненты
  - ClassLoader
  - Bytecode Verifier
  - Привилегированный код в системных классах Java (например sun.awt.*)

Количество найденных уязвимостей в Java (Oracle)



CVE Report

Уязвимости по типам в Java (Oracle)



CVE Report

CVE-2012-0507 (Caberp)

Платформы:

- Oracle JDK/JRE 1.7.0
- Oracle JDK/JRE 1.6.0_27 и ниже
- Sun JDK/JRE 1.5.0_35 и ниже

Описание: Ошибка в реализации *AtomicReferenceArray* при работе с массивами, позволяющая выполнить произвольный код за пределами песочницы

Metasploit URL: http://goo.gl/q51FOp

CVE-2012-1723

Платформы:

- Oracle JDK/JRE 1.7.0_4 и ниже,
- Oracle JDK/JRE 1.6.0_32 и ниже
- Sun JDK/JRE 1.5.0_35 и ниже
- Sun JDK/JRE 1.4.2_37 и ниже

Описание: Ошибка в реализации HotSpot, позволяющая выполнить произвольный код за пределами песочницы

Metasploit URL: http://goo.gl/6GL7p1

Разбор Malware CVE-2012-1723

ВНИМАНИЕ! ДАННЫЙ РАЗДЕЛ СОДЕРЖИТ СЦЕНЫ НАСИЛИЯ НАД HOTSPOT И РАСЧЛЕНЕНКУ БАЙТКОДА!

Разбор malware на базе CVE-2012-1723

- Анатомия и способ распространения
- Подробности про CVE-2012-1723
- Способы сокрытия логики работы
- Анализ байткода

Анатомия и способ распространения

- Модуль поиска уязвимостей
  - При попадании пользователя на сайт злоумышленника, определяется его версия Java и активируется необходимый Exploit
- Exploit (Trojan.Dropper)
  - Основная цель нарушить модель безопасности песочницы, скачать и запустить вредоносный нативный код
- Вредоносный код (Payload)
- Центр управления (CC)

Способы сокрытия логики работы

- Переименование классов/методов/ переменных
- Манипуляции с константами
- Вызов методов через Reflection
- Динамическая загрузка байткода loadClass(..)
- Шифрование строковой информации

Шифрование строковой информации

- Загрузка класса по имени

```
class.forName(“SomeSecretClass”) ->
class.forName(decryptFunc(“qelr2342432w3qdz”))
```

- Поиск метода по имени

```
class.getMethod(“verySecretMethod”) ->
class.getMethod(decryptFunc(“werwr42312e3”))
```

- Загрузка секретного ключа или массива байт (спрятанный байткод)

Анализ байткода

- Самый эффективный инструмент javap – дизассемблер байткода, идущий в составе JDK
- Детектирование использования уязвимости
- Снятие шифрования строк

Детектирование использования уязвимости

- Дизассеблируем с помощью вызова

```
javap -c -v BadClass
```

- CVE-2012-1723 можно обнаружить по “ненормальному” количество полей одного типа в классе, не менее 100, циклу по разогреву HotSpot + методом с последовательными инструкциями PUTSTATIC/GETSTATIC/PUTFIELD/GETFIELD

Детектирование использования уязвимости

```
public class pas {
static java.lang.ClassLoader fob;
sax fog1;
sax fog2;
….
sax fog95;
sax fog96;
sax fog97;
…
}
```

“Ненормальное” количество полей одного типа в классе + поле жертва fob

Детектирование использования уязвимости

```
sax gat(java.lang.ClassLoader);
Code:
0: aload_1
1: ifnonnull 6
4: aconst_null
5: areturn
6: getstatic #2 // Field java/lang/System.out:Ljava/io/PrintStream;
9: getstatic #3 // Field fob:Ljava/lang/ClassLoader;
12: invokevirtual #4 // Method java/io/PrintStream.print:(Ljava/lang/Object;)V
15: aload_0
16: nop
17: aload_1
18: putfield #3 // Field fob:Ljava/lang/ClassLoader;
21: aload_0
...
1181: getfield #101 // Field fog97:Lsax;
1184: areturn
1185: aconst_null
1186: areturn
}
```

Метод эксплуатации уязвимости (GETSTATIC/PUTFIELD/GETFIELD)

Снятие шифрования строк

Оригинальный код

```
System.out.println("HelloWorld");
```

Байткод

```
ldc             #3; //String HelloWorld
invokevirtual   #4;//java/io/PrintStream.println:(Ljava/lang/String;)V
```

Байткод с зашифрованной строкой

```
ldc             #3; //String CryptedString
invokestatic    #4; //Method decrypt:(Ljava/lang/String;)Ljava/lang/String;
invokevirtual   #5; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Снятие шифрования строк

Основная идея: найти в байткоде вызовы функции дешифрования строки, и потом модифицировать существующий байткод, убрав вызовы функции дешифрования и вставив оригинальные строки.

Берем ASM (http://asm.ow2.org) и пишем прототип

Снятие шифрования строк

```
class Decryptor extends ClassVisitor implements Opcodes {


public MethodVisitor visitMethod(int access, String name, String desc,
String signature, String[] exceptions) {
   MethodVisitor mv = cv.visitMethod(access, name, desc, signature, exceptions);
   return new MethodDecryptor(mv, decMethod);
}


public MethodDecryptor(MethodVisitor mv, Method decMethod) {
    super(ASM4, mv);
    this.decMethod = decMethod;
    decClassName = decMethod.getDeclaringClass().getName().replace(".","/");
    decMethodName = decMethod.getName();
}
```

Снятие шифрования строк

```
public void visitMethodInsn(int opcode, String owner, String name, String desc)
{
  // skipping decryption function
 if (!(decClassName.equals(owner) && decMethodName.equals(name) && "(Ljava/lang/
                    String;)Ljava/lang/String;".equals(desc))) {
        super.visitMethodInsn(opcode, owner, name, desc);
 }
}

public void visitLdcInsn(Object o) {
    if (o instanceof String) {
        try {
            // decrypt string
            o = decMethod.invoke(null, new Object[]{o});
            } catch (Exception ex) {}
    }
    super.visitLdcInsn(o);
}
```

Развитие механизмов безопасности Java

Развитие механизмов безопасности Java

- Оперативное реагирование на найденные уязвимости
  - Critical Patch Updates (CPU)
  - Security Alerts (RSS)

Развитие механизмов безопасности Java

- Планомерный ввод новых security механизмов

Развитие механизмов безопасности Java

- Защита Java-апплетов
  - Блокировка неподписанные и самоподписанных апплетов на уровне с минимальными привилегиями
  - Автоматическое обновление списка отозванных сертификатов
  - Расширенные политики безопасности установки апплетов
  - Новые предупреждающие диалоги при работе с апплетом

Развитие механизмов безопасности Java

- Server JRE
  - Базируется на JDK
  - Только x64
  - Версии для Java 7/Java8
  - Убрана поддержка deployment stack (Applets, JavaWebStart, JavaFX)
  - Удалены некоторые клиентские библиотеки и JavaFx

Контакты

- Email: dudarev@licel.ru
- Twitter: @MikhailDudarev
- Web: http://licelus.com